Multilevel Skeletonization Using Local Separators

J. Andreas Bærentzen
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Rasmus Emil Christensen
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Emil Toftegaard Gæde
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Eva Rotenberg
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Abstract

In this paper we give a new, efficient algorithm for computing curve skeletons, based on local separators. Our efficiency stems from a multilevel approach, where we solve small problems across levels of detail and combine these in order to quickly obtain a skeleton. We do this in a highly modular fashion, ensuring complete flexibility in adapting the algorithm for specific types of input or for otherwise targeting specific applications.

Separator based skeletonization was first proposed by Bærentzen and Rotenberg in [ACM Tran. Graphics'21], showing high quality output at the cost of running times which become prohibitive for large inputs. Our new approach retains the high quality output, and applicability to any spatially embedded graph, while being orders of magnitude faster for all practical purposes.

We test our skeletonization algorithm for efficiency and quality in practice, comparing it to local separator skeletonization on the University of Groningen Skeletonization Benchmark [Telea'16].

2012 ACM Subject Classification Computing methodologies → Computer graphics; Theory of computation → Computational geometry; Software and its engineering → Software design engineering

Keywords and phrases Algorithm engineering, experimentation and implementation, shape skeletonization, curve skeletons, multilevel algorithm

Digital Object Identifier 10.4230/LIPIcs.SoCG.2023.13

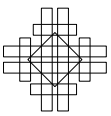
Related Version *Full Version*: `https://arxiv.org/abs/2303.07210`

Supplementary Material
Software: `https://github.com/janba/GEL` [10]
archived at `swh:1:dir:91f125aa9a06d7adf992cdb11ca2108d86acdefe`
Software (Supplementary repo for testing and additional variations): `https://github.com/Sgelet/GEL`, archived at `swh:1:dir:09aadd7e8f82cc8dfdbbc72ea71e56e65a6e0cfc`

Funding *Emil Toftegaard Gæde* and *Eva Rotenberg*: supported by Eva Rotenberg's Carlsberg Foundation Young Researcher Fellowship CF21-0302 - "Graph Algorithms with Geometric Applications".
J. Andreas Bærentzen: Partially supported by the Villum Foundation through Villum Investigator Project InnoTop.

1 Introduction

A curve skeleton is a compact simplified representation of a shape, consisting only of curves. The act of *skeletonization*, in this context, is the computation of such a curve skeleton for a given input. For the remainder of this paper *skeleton* refers exclusively to curve skeletons. Various fields, including feature extraction, visualisation and medical imaging, care not only



about shapes and objects, but also about their structures and features. In applications such as shape matching, the skeleton acts as a simplified representation of an object, allowing for reduced computation cost [8], whereas in virtual navigation the curve skeleton can act as a collision free navigational structure [24, 30].

The broad areas of application, and the different roles that skeletons play, lead to differing interpretations of exactly what the skeleton is. Although no widely agreed upon definition of skeletons exist, work has been done on narrowing down desirable properties of skeletons in the general case [12].

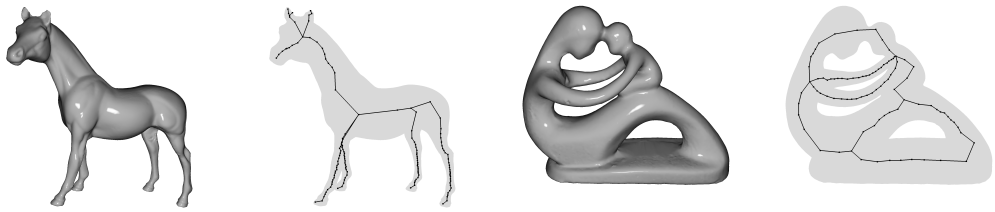


Figure 1 Shaded renders of triangle meshes and skeletons obtained by our algorithm.

Instead of giving a formal definition, we will base our work on the evocative if imprecise definition of skeletons as simplified curve representations of the underlying structure and topology. In Figure 1 we show skeletons of various input, to exemplify our definition.

Many different approaches to skeletonization exist [27], such as computing and pruning the medial surface [13], computing mean curvature flow [26] or contracting meshes [20]. In a recent paper A. Bærentzen and E. Rotenberg present a new algorithm that bases itself on computing *local separators* [5]. We refer to this algorithm as the local separator skeletonization algorithm, *LSS*. This approach has the benefit that it requires only that the input be given as a spatially embedded graph, rather than a specific shape representation. This makes the method applicable to a wide variety of inputs, such as meshes, voxel grids or even input that does not necessarily represent a shape. In addition, the skeletons that it generates are of high quality, capturing features that contractive methods tend to miss. However, the algorithm is also computationally expensive.

In this paper we present a multilevel algorithm for computing curve skeletons that we obtain by adapting LSS to a multilevel framework. Below, we start with some preliminaries and then present an overview of our contributions. Next, after a discussion of related work, we describe our approach to graph coarsening, projecting separators onto finer level graphs, and, finally, the multilevel skeletonization algorithm that builds on these components. We provide analyses of the algorithms in the paper and we test our work on a skeletonization benchmark. Our results show that our algorithm is orders of magnitude faster than that proposed by Bærentzen and Rotenberg while producing skeletons of comparable quality.

1.1 Preliminaries

We consider the discrete skeletonization problem, where both the input and output is represented by spatially embedded undirected graphs. Formally, we consider skeletonization of a graph $G = (V, E)$ where each vertex is associated with a geometric position $p_{v \in V} \in \mathbb{R}^3$. Note that we make no other assumptions about the graph, such as whether it is sampled from the surface of a manifold, created from a point cloud, or otherwise.

In graph theory a *vertex separator* is a set of vertices whose removal disconnects the graph. In [5], this notion is extended to *local separators*, defined as a subset of vertices, $S \subset V$, that is a vertex separator of the subgraph induced by the closed neighbourhood of S. Likewise, the notion of a minimal local separator is defined as a local separator that is a minimal vertex separator of the subgraph induced by the closed neighbourhood. Intuitively, we cannot remove a vertex from a minimal local separator without the remaining set ceasing to be a local separator. For the rest of this paper, the term *separator* means local separator.

1.2 Contributions

The LSS algorithm computes skeletons through a three-phased approach. A large number of minimal local separators is computed, the minimal separators are selected using a greedy packing method, and, lastly, the skeleton is extracted from the packed set of minimal separators. A visualisation of these phases can be seen in Figure 2.

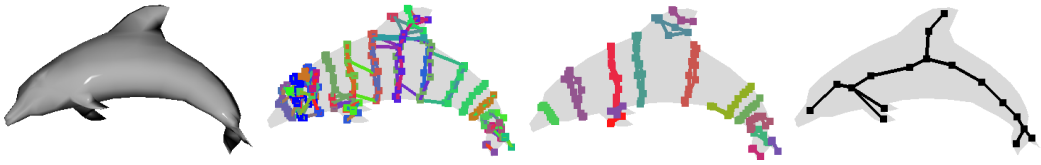


Figure 2 Visualisation of the three phases of the LSS algorithm. From left to right: A shaded render of the input, a number of computed minimal separators, a non-overlapping subset of the separators, and the resulting skeleton after extraction.

As the algorithms for the first two phases play an intrinsic role in our algorithm, we give a brief description of these.

Computing local separators is done through a two-step process. First a region growing approach is used to find a local separator. A vertex is picked, and we iteratively add to the separator an adjacent vertex and check if the neighbourhood is disconnected. We refer to this as *growing* a separator. Once a local separator has been found, it is heuristically minimised by removing vertices that would not destroy the separator. We refer to this as *shrinking* a separator.

Because the running time of LSS is often dominated by the search for local separators, a sampling scheme is used to reduce computation. According to the scheme, vertices are selected for separator computation with probability 2^{-x}, where x is the number of previously computed separators that contain that vertex.

Unfortunately, sampling only addresses the number of separators that need to be computed and not the time it takes to compute each separator. In this paper we address the latter issue using a multilevel approach. Specifically, we find separators on coarser versions of the graph and project them back up onto the original graph. Importantly, this allows us to set a *patience threshold* for the amount of computation that should be used to find a separator from a given vertex. When the threshold is exceeded we stop the search relying on a separator containing the given vertex to be found on a coarser level.

1.3 Related Work

Skeletonization, in terms of computing curve skeletons, is a diverse field not only in terms of interpretations of skeletons, but also in the algorithmic approaches. Several classifications of algorithms exist [12, 27], based on underlying traits of the algorithms.

The interpretation that the curve skeleton should lie on the medial surface, gives rise to methods that, in a sense, extract a curve skeleton from the medial surface of the input [13, 29, 19, 25, 32]. Since the medial surface is highly sensitive to noise, so are the skeletons generated by these methods.

A class of algorithms that are resilient to noise are the contractive methods, based on the concept of reducing the volume and surface area of the input until a skeleton is found [31, 4, 26]. In their simplest forms these algorithms require that the input be manifold, however it is possible to extend to other types of input [11, 18].

A related notion for shape analysis is that of Reeb graphs [7, 6]. These can be used for skeletonization, lending themselves to a topologically driven class of algorithms [22, 23, 14]. The resulting skeletons depend on a parameter, giving some flexibility in targeting specific properties of the output, but also requiring great care in the choice of the parameter.

In addition there are algorithms that fit into classifications not presented here [20, 15, 2].

A very successful heuristic approach to the NP-complete problem of graph partitioning is that of multilevel algorithms [9]. Although the problem considered is different, we employ a similar multilevel scheme for vastly improving practical performance. Such multilevel schemes have been extensively studied [16, 21, 1].

2 The Multilevel Framework

In its most general sense, the multilevel framework is a heuristic approach that aims to solve a problem by obtaining a solution to a smaller problem.

Initially, a series of increasingly simplified approximations of the input is generated. We call this the *coarsening* phase, and the series of simplifications we call *levels*. Since the last level is small, computing a solution is much faster. In graph partitioning literature, this is called the *partitioning* phase; however, we will consider it in terms of solving a *restricted* problem. Then, the solution found on the last level is transformed into a solution on the input through *uncoarsening*. This process is also sometimes called *projection and refinement*, since uncoarsening from one level to the previous is often done by projecting onto the previous level, and then employing some refinement process to improve the solution according to some heuristic.

By design, the multilevel framework is highly flexible. Various coarsening schemes can be used, that may prioritise preserving different properties of the input when simplifying. The restricted problem can be solved by any reasonable approach, and the refinement strategies can be adapted to suit the application.

Our algorithm works by first coarsening the input into several levels of decreasing resolution. The details of this coarsening is described in Section 2.1. Once the hierarchy of graphs has been generated, we do a restricted search for local separators on each level of resolution. The details are covered in Section 2.2, but the intuition is that searching for large local separators is slow in practice, and by restricting our search we save computation. Since the separators found are small, this does however also mean that we are only able to capture small features of the structure.

Since small features obtained on low resolution can represent large features on the original input, we obtain separators capturing features of varying sizes by searching for separators across every level.

By projection and refinement, see Section 2.3, we then transform the minimal local separators found across the levels into minimal local separators on the input. These can then be packed and extracted by the approach of LSS. The procedure is visualised in Figure 3.

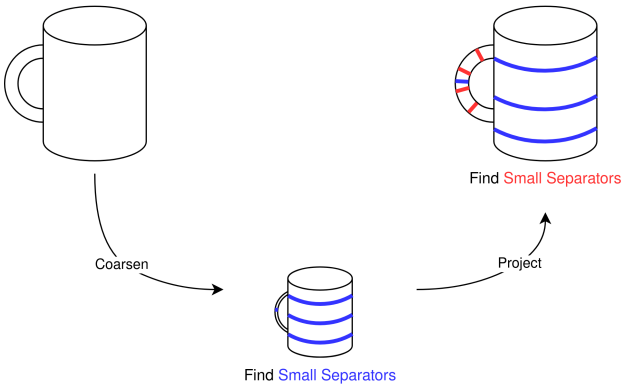


Figure 3 Visualisation of the multilevel skeletonization approach. A solid cylinder with a handle is coarsened until it is of small size. A number of small local separators are found (shown in blue), and then projected back to the original input. Searching for small local separators again yields the separators around the handle (shown in red), but separators are too large at this level to be discovered around the cylinder. We combine the separators to obtain a general solution.

2.1 Coarsening

Given as input a graph, $G = (V, E)$, we construct a sequence of increasingly simplified graphs, $G_0, G_1, \ldots, G_l$ s.t. $G_0 \succ G_1 \succ \ldots \succ G_l$ where $l = O(\log n)$ and $G_i \succ G_j$ denotes that G_j is a minor of G_i, and $G_i = (V_i, E_i)$. Moreover $G_0 = G$ and $\forall i \in [0, l), |V_i| \geq 2|V_{i+1}|$.

We do this by a matching contraction scheme, in which we repeatedly construct and contract maximal matchings. Various approaches to such coarsening schemes exist in literature [21], and from these, we choose to consider *light edge matching*.

To construct G_{i+1} from G_i, greedily find a maximal matching and contract it. Such a matching can be constructed in $O(|E_i|)$ time by visiting vertices in a random order, matching them to an unmatched neighbour of smallest euclidean distance. We repeat this procedure until the number of vertices has been at least halved.

In Figure 4 we show some of the graphs obtained during coarsening of a triangle mesh resembling a statue of Neptune.

Figure 4 A series of increasingly simplified approximations of `neptune.ply`, from the Groningen Skeletonization Benchmark, obtained through light edge matching contraction.

Note that by contraction we always preserve the number of connected components. This is one of the homotopy-preserving properties of the algorithm.

Theoretical Analysis

In the worst case, we may spend $O(|V_i|)$ rounds of contraction in order to reach the desired number of vertices. This is a well known problem of matching contraction schemes on general graphs, but graphs obtained from the world of geometry tend to take a small number of rounds to contract [1].

A general bound on the time spent on the coarsening phase is then $\sum_{i=0}^{l}(|V_i||E_i|) = |E|\sum_{i=0}^{l}|V_i| = O(|V||E|)$. For graphs that are contracted in a constant number of rounds we get $\sum_{i=0}^{l}|E_i|$ and if we furthermore have $|E_i| = O(|V_i|)$, as is the case for triangle meshes and voxel grids, the bound becomes $O(|V|)$.

2.2 Restricted Separator Search

For a given connected set of vertices, V', we refer to the subgraph induced by vertices adjacent to V', that are not in V' themselves, as the *front* of V' and denote it $F(V')$.

A region growing based approach to computing local separators is given in [5], where a separator, Σ, is iteratively grown until $F(\Sigma)$ is disconnected. The approach uses an enclosing ball around the vertices of Σ to guide what vertex of $F(\Sigma)$ is added next, and the connectivity of $F(\Sigma)$ is then checked by traversal. As noted by the authors, it is possible to improve performance of the search by using a dynamic connectivity data structure to maintain the front, so that a traversal in every iteration is avoided.

In addition to adapting the algorithm to use a dynamic connectivity data structure, we will also restrict the number of iterations the search performs. Given a vertex, v, set $\Sigma_0 = \emptyset, F_0 = \{v\}$, and then iteratively construct $\Sigma_i = \Sigma_{i-1} \cup \{v_i \in F(\Sigma_{i-1})\}$, where v_i is the closest neighbour of the front to an enclosing sphere around Σ_{i-1}. Maintain $F(\Sigma_i)$, update the enclosing sphere and repeat until $F(\Sigma_i)$ is disconnected or empty, or $|\Sigma_i|$ exceeds a threshold value. Pseudocode for this restricted separator search is shown in Algorithm 1.

Theoretical Analysis

We analyse the complexity of this restricted separator search in terms of the graph $G' = \Sigma \cup F(\Sigma)$ with n' vertices and m' edges, using the dynamic connectivity data structure of Holm, de Lichtenberg and Thorup with updates in amortized $O(\log^2 n')$ time [17]. Since the size of the separator is at most α and we add one vertex each iteration, we use at most α iterations selecting the closest vertex from the front, updating the bounding sphere and maintaining the dynamic connectivity structure. Selecting the closest vertex is done naively with a scan through the front, taking $O(n')$ time and updating the bounding sphere takes $O(1)$ time each iteration. This gives a running time of $O(\alpha n')$. Each edge in the dynamic connectivity structure is inserted and removed at most once, with each operation taking $O(\log^2 n')$ amortized time, totalling $O(m' \log^2 n')$. The total running time is then $O(\alpha n' + m' \log^2 n')$.

In the general case we give no better worst case bound than $O(\alpha|V| + |E|\log^2|V|)$. For graphs of bounded maximum degree we can bound the size of the front. Let Δ be the maximum degree of G, then $n' = O(|\Sigma|\Delta) = O(\alpha\Delta)$ and $m' = O(\alpha\Delta^2)$. This gives a time of $O(\alpha^2\Delta + \alpha\Delta^2 \log^2(\alpha\Delta))$. In addition if we choose α to be a small constant, the bound is further improved to $O(\Delta^2 \log^2 \Delta)$. For graphs where $\Delta = O(1)$ as for voxel grids or knn-graphs, the search then becomes $O(1)$. Note that for this bound to be applicable across the entirety of the algorithm, the degree needs to remain bounded through coarsening.

Algorithm 1 Restricted Separator Search

Given a spatially embedded graph, G, a starting vertex, v_0, and a thresholding value, α, search for a separator of size at most α and return it, or $\emptyset$ if failure. Here ϵ is a small constant to prevent division by zero.

```
RESTRICTED-SEPARATOR-SEARCH(G, v_0, α):
    Σ = ∅
    F = ({v_0}, ∅)
    c = p_{v_0}
    i = 0
    r = 0
    repeat
        v = arg min_{f∈V(F)} ‖c − p_f‖          // Scan front for closest vertex
        if ‖c − p_v‖ > r then
            r = ½(r + ‖c − p_v‖)                 // Update the sphere
            c = p_v + r/(ε+‖c−p_v‖)(c − p_v)
        Σ = Σ ∪ {v}
        REMOVE(F, v)
        for (x, y) ∈ E(NEIGHBOURHOOD(G,v) − Σ) do
            CONNECT(F, x, y)                     // Maintain the front of Σ
        i = i + 1
    until NUMBER-OF-COMPONENTS(F) > 1 or i = α
    if NUMBER-OF-COMPONENTS(F) = 1 then
        return ∅
    return Σ
```

2.3 Projection and Refinement

For projecting separators to graphs of higher levels of detail, we employ a simple uncoarsening technique. By storing information about what vertices were contracted during coarsening, we can reverse the contractions that gave rise to the vertices of a given separator. Note however that a separator that has been projected in such a way is not guaranteed to be minimal.

The simplest refinement scheme is thus one that uses the algorithm for minimising separators as in LSS. The minimising algorithm is a heuristic approach that seeks to minimise a separator such that the structure becomes that of a thin band. When used on separators that are obtained through projection, there is not necessarily much room for choice. Therefore we consider a variation of our refinement scheme, we thus choose to "thicken" the separators after projection, by adding the adjacent vertices if it would not destroy the separator. This gives the heuristic minimisation more options for creating separators of shorter length, as visualised in Figure 5.

Projecting a separator can be done in linear time proportional to the size of the resulting separator, while the minimisation in worst case takes quadratic time (see the full version).

After processing each separator in this way, we obtain a set of minimal separators for the current level. If we accumulate separators indiscriminately, we will spend time projecting and refining separators that will ultimately be discarded due to overlapping. If we perform set packing on every level, we are going to be too eager in our efforts, discarding things that might not overlap once projected further. Intuitively, we would like to only discard separators if there is a large overlap.

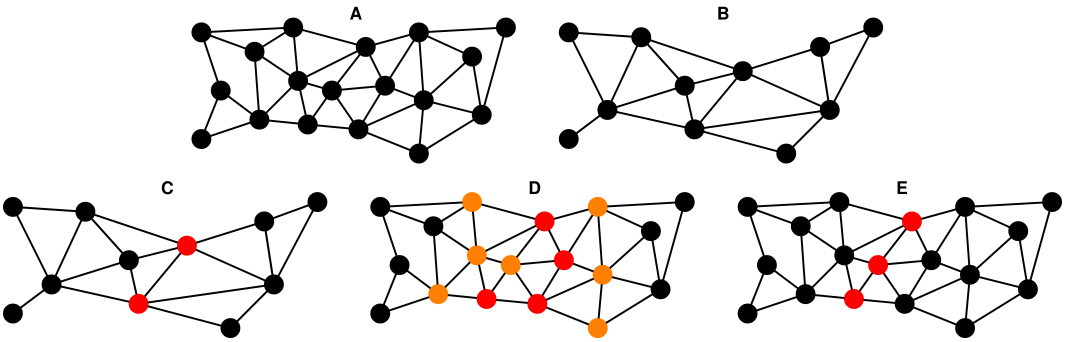


Figure 5 A separator undergoing expansion as part of refinement. (A) shows an input, (B) a coarsened representation, (C) a computed separator denoted by red vertices, (D) the projected separator denoted by red vertices and the added vertices denoted by orange, (E) shows the separator obtained by minimising the thickened separator.

To do this, we associate with each vertex, v, of every graph, G_i, a capacity, c_v^i, equal to the sum of capacities of vertices contracted to obtain it. For vertices of G_0 we define the capacities as 1, formally $\forall v \in V(G_0), c_v^0 = 1$. In this way, the capacity of a given vertex is the number of vertices of the original input that it represents.

We then modify the greedy set packing algorithm of LSS, so that we include a separator iff it would not cause any vertex to exceed its capacity. Since the capacities of G_0 are 1, this packing is equivalent to the original when applied to the highest level of resolution, and thus we will still have a non-overlapping set of separators at the end.

Note however that this packing allows for duplicates to persist through packing, essentially reducing the capacities of vertices while providing no valuable information. To counteract this, we perform a filtering step using hashing to rid duplicates prior to the packing procedure. Filtering and packing in this way takes time linear in the sum of sizes of separators in the set.

2.4 The Multilevel Skeletonization Algorithm

With the details of the phases in place, we can then combine these to construct the multilevel skeletonization algorithm. Given a spatially embedded input graph, $G = (V, E)$, and a threshold value α, we construct a curve skeleton by the following:

Generate G_i from G_{i-1} by coarsening, until $|G_l| \leq \alpha$ for some l. This generates the sequence of graphs of decreasing resolution $G_0, G_1, \ldots, G_l$ where $l = O(\log |V|)$ and $\sum_{i=0}^{l} |V_i| = O(|V|)$.

Then, starting at the lowest resolution, G_l, find restricted separators. We do this by the restricted separator search, starting at each vertex with probability 2^{-x}, where x is the number of currently computed separators containing that vertex, using α as the restriction on the size of the search. After computing the separators for a level, we perform capacity packing, and then we project the computed separators to the next level and refine them. This process is repeated for every level until we arrive at the original graph. At this point, after performing capacity packing, we obtain a non-overlapping set of minimal separators from which we extract the skeleton, using the extraction procedure of LSS [5]. Pseudocode for this algorithm can be seen in Algorithm 2.

Algorithm 2 Multilevel Skeletonization
Given a spatially embedded graph, G, and a thresholding value, α, compute a curve skeleton.

```
MULTILEVEL-SKELETONIZATION(G, α):
    G_0 = G
    l = 0
    repeat                                                  // Coarsening phase
        l = l + 1
        G_l = COARSEN(G_{l-1})
    until |V(G_l)| ≤ α
    S = ∅                                  // Maintain set of minimal separators
    for i = l to 0 do                              // From low to high resolution
        S' = ∅
        for s ∈ S do                  // Project and refine from previous levels
            S' = S' ∪ PROJECT-REFINE(s)
        S = S'
        for v ∈ V(G_i) with probability 2^{-x(v)} do      // Search on this level
            s = RESTRICTED-SEPARATOR-SEARCH(G_i, v, α)
            S = S ∪ {MINIMISE-SEPARATOR(s)}
        S = CAPACITY-PACK(S)
    return EXTRACT-SKELETON(G, S)
```

Theoretical Analysis

For completeness' sake we consider then the complexity of the algorithm. Recall that the coarsening phase in the general worst case takes $O(|V||E|)$ time, but for not too irregular input takes $O(|V|)$ time. We then perform a restricted separator search from each vertex across every level, which is $O(\alpha|V|^2 + |V||E|\log^2|V|)$ in the general worst case, but $O(\alpha|V|)$ for graphs that retain constant bounded degree through coarsening. We consider then the time a single separator contributes to the total when expanding, filtering and packing. These operations are linear in the size of the separator on each level, which is worst case $O(|V_i|)$ on level i. This totals $\sum_{i=0}^{l} O(|V_i|) = O(|V|)$ for a single separator across all levels. Minimizing a single separator takes worst case $O(|V_i|^2)$ on level i, which for a single separator contributes $\sum_{i=0}^{l} O(|V_i|^2) = O(|V|^2)$ across all levels. We also perform packing on each level, linear in the sum of sizes of separators, which in total takes $\sum_{i=0}^{l} O(|V_i|^2) = O(|V|^2)$ time. The general worst case bound then becomes $O(|V|^3 + |V||E|\log^2|V|)$, which is an improvement over LSS. It is worth mentioning however, that in practice the running time for both LSS and our algorithm is heavily dominated by the search for separators, and that theoretically expensive procedures, such as minimisation, make up only a small fraction of the running time.

3 Experiments

In this work, our main objective was to make an algorithm that produces the same quality of skeletons as LSS [5], only with improved running times, using new algorithmic ideas and algorithm engineering. As we will show in this section, the improvements to practical running times are very satisfactory.

As for quality, it is our overall assessment that the quality has not been compromised by the speed-up.

There is, however, no standard for how skeletons should be compared. In [27] it is remarked that quantifying the quality of skeletons is an open challenge, but we shall instead compare ourselves only to skeletons obtained by LSS, to quantify the deviation obtained by employing the multilevel approach.

To do this, we will measure a number of metrics, namely the number of vertices in the skeleton, the number of leaf nodes, branch nodes, chordless cycles which estimates the genus of the input, and the directed Hausdorff distance in both directions. For our comparisons, it is the relationship between directed Hausdorff distances that matter, rather than the magnitudes. A high distance from an LSS skeleton to our skeleton, with a low distance the other way, could indicate that LSS captures a feature that our skeleton does not. Likewise for the inverse, which might indicate that we are capturing a feature that LSS does not deem to exist. We give our Hausdorff distances divided by the radius of a bounding sphere, to reduce influence from the differing scales of input.

We run our tests on the Groningen Skeletonization Benchmark [28], consisting of several triangle meshes of varying structure. The test are executed on HPC Cluster nodes with Xeon Gold 6226R (2.90GHz) CPUs, using 8 cores of a single CPU for each test. For running time measurements, tests are run three times, and the median value is reported.

For comparisons we examine three algorithms, namely the local separator skeletonization algorithm (LSS) [5], our multilevel algorithm using light edge matchings, described in Section 2.1, as contraction scheme (LEM) as well as with light edge matchings and thickened separators, described in Section 2.3, as the refinement scheme (LEMTS).

We note that the variation of LSS with which we compare our algorithms also includes usage of a dynamic connectivity structure, so that the search procedures are identical up to the threshold parameter.

Implementation

Our implementation is written in C++, built into the GEL library, and made publicly available [10]. This is the same library that contains LSS, and as such our algorithms use the same underlying data structures and subroutines. All programs are compiled using `-O3` optimisation flags. Details regarding the dynamic connectivity structure are given in the full version. We run the restricted searches in parallel internally on each level, using a simple fork-join pattern, identical to that of LSS. To account for the multilevel structure of our algorithm, we then pack using a single thread, project using at most two threads and repeat the pattern for the next level. The decision to use only two threads for projection stems from the fact that the overhead associated with spinning up threads quickly outweighs the benefits of parallel projection since there are often few separators after packing.

3.1 Results

Here we present our results in terms of measurements on the Groningen Skeletonization Benchmark. Initially we argue for our choice of α, showing how the threshold impacts both skeleton quality as well as running time. We then present a number of results relating to the skeletons themselves to showcase the quality of output. We then present our measurements of running times, as well as discuss interesting observations.

Additional measurements are presented in the full version.

The Right Amount of Patience – Determining a Threshold Value α

To show the effects of α on the running time, we consider a small test suite using subdivisions of a mesh to generate various sizes of input. This is done to ensure a similar underlying structure throughout the test. We test our multilevel algorithm for $\alpha = 8, 16, 32, 64, 128$ (see Figure 6). Not surprisingly, a lower threshold leads to a lower running time.

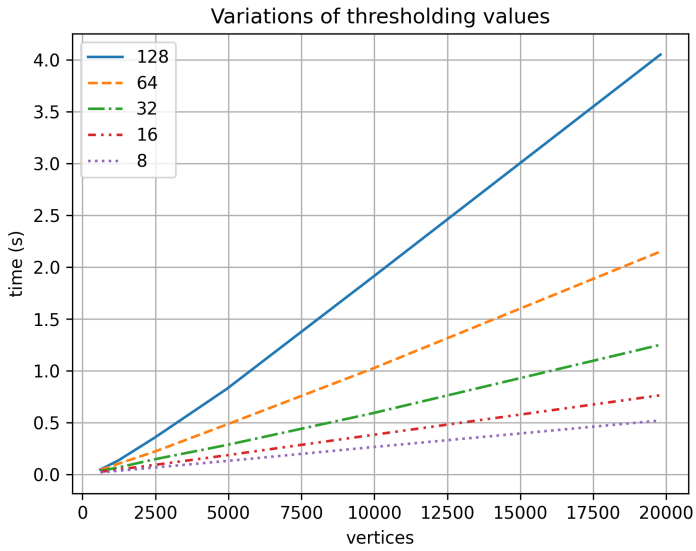


Figure 6 Running time measurements of varying values of α on subdivisions of a mesh.

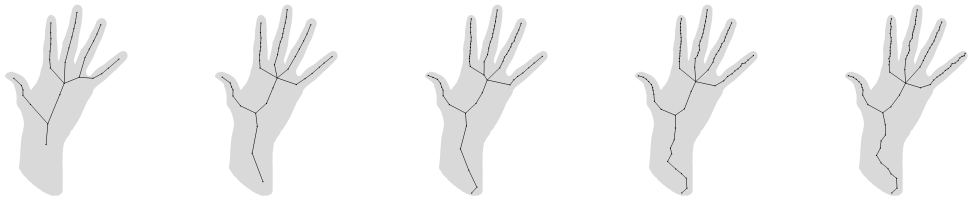


Figure 7 Skeletons found on `human_hand.ply` for increasing patience thresholds. From left to right: $\alpha = 8, 16, 32, 64, 128$.

However, there may be a trade-off between skeleton quality and threshold value, which we explore qualitatively.

We visually examine the skeletons generated for our choices of α on `human_hand.ply`, as seen in Figure 7. For very low values of α, the skeletons have few curves in areas that are relatively thick. With a low threshold, separators must be found on lower resolutions, which in turn means that few separators can be found. As we progress to higher values of α, the level of detail of the skeleton rises, up to a certain point. Intuitively, if the threshold is high enough that the details can be captured on the higher levels of detail, then we gain nothing from the lower resolution levels.

It could be argued that $\alpha = 16$ or $\alpha = 32$ generates the most visually appealing skeletons for this particular input, however we find that $\alpha = 64$ offers the best trade-off for running time on other examined input such as that shown in Figure 8.

Thus, we run the remainder of our tests using $\alpha = 64$.

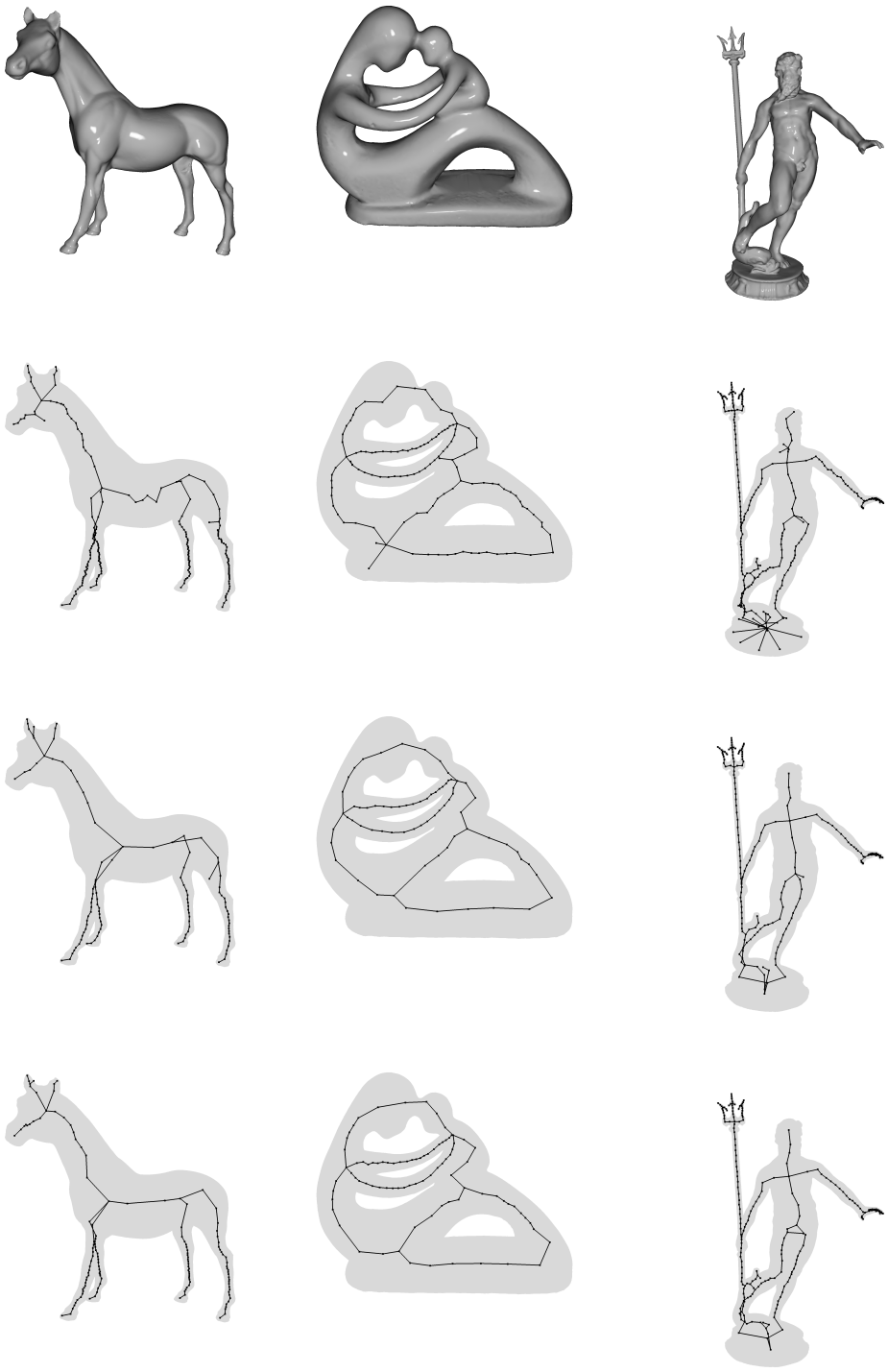


Figure 8 Each column indicates a different input, with each row showcasing a different method. From top to bottom: shaded renders of the input, skeletons obtained by LSS, skeletons obtained by LEM, skeletons obtained by LEMTS.

Skeleton Quality

In Figure 8 we show some of the skeletons obtained by LSS, LEM and LEMTS. Note that both LEM and LEMTS appear smoother, while also reaching the features that LSS finds in many cases. In addition, it seems that for these inputs our methods find less spurious features, giving a cleaner result. When comparing LEM and LEMTS the differences are subtle. On `horse.ply` it can be seen that the vertex where the front legs meet the body is positioned further to the left. This is due to LEMTS having a denser skeleton. On the other hand, LEMTS seems to not capture the structure of the groin area of `neptune.ply` as well as LEM. Although it appears as if LEMTS finds a cycle, this is not actually the case. The skeletal branch is, however, not positioned as one would expect.

Table 1 Excerpt of measurements on skeletons. The metrics denoted by Δ are relative to the skeletons of LSS, with negative values implying that LSS has more vertices, leafs, branches etc. Here $H(A, B)$ denotes the directed Hausdorff distance between A and B, divided by the radius of a bounding sphere, and $*$ denotes skeletons generated by our multilevel algorithms.

input	algorithm	Δvertices	Δleafs	Δbranches	Δgenus	$H(LSS, *)$	$H(*, LSS)$
19465	LEM	-563	-3	-35	2	0.0437028	0.0415616
	LEMTS	-416	11	-24	3	0.0355444	0.0382796
fertility	LEM	-46	-3	-1	0	0.273804	0.0874419
	LEMTS	-30	-3	-1	0	0.240033	0.165239
happy4	LEM	-589	-409	-21	-100	0.199295	0.0887277
	LEMTS	-560	-384	-22	-99	0.160472	0.100934
horse	LEM	-95	-1	-1	0	0.112222	0.111066
	LEMTS	-66	-2	-2	0	0.0883287	0.0851157
neptune	LEM	-73	-16	-5	0	0.200924	0.0508065
	LEMTS	-51	-15	-1	0	0.225756	0.0516766

In addition, we also showcase a small excerpt of measurements from the full version, which can be seen in Table 1. Here it is clear that LEM and LEMTS produce slightly simpler skeletons with fewer vertices, leaves, and branches. However, from visual inspection of the models it is clear that (at least for the models in the table) the missing details in the skeleton correspond to features which are so subtle that the skeletal details might be considered spurious. For all inputs of the benchmark except `happy4.ply`, there is little deviation in the genus compared to LSS.

For context on the strange genus found on `happy4.ply`, we show the generated skeletons in Figure 9. Of note is that the mesh has several missing patches, which seems to cause spurious small separators to be found on all of the local separator based methods. We consider this an error case for all of the methods examined.

Running Time

Although the running time of local separator skeletonization methods depends very much on the search for separators, which in turn depends on the structure of the input, we give the running times of the examined methods in Figure 10 as a function of the number of vertices in the input, over the entirety of the Groningen Skeletonization Benchmark [28].

Remarkably, we find that the multilevel algorithm not only outperforms LSS by several orders of magnitude, but also that it seems to be less dependant on the underlying structure of the triangle meshes, giving what appears to be a slightly superlinear curve. This effect is even

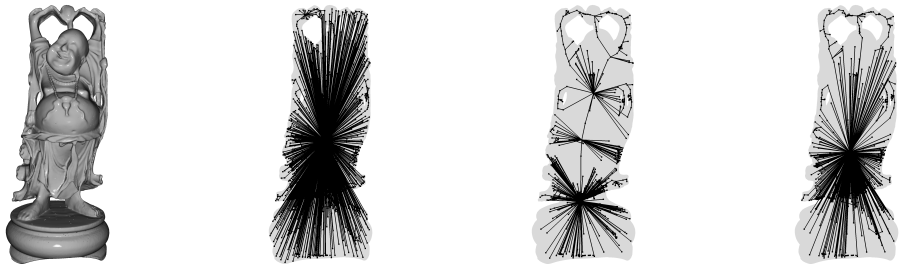


Figure 9 A triangle mesh with a large number of missing patches on the surface, `happy4.ply`, resulting in erroneous output for LSS, LEM, and LEMTS.

more pronounced when considering only the time to search for separators. Under assumptions about the degree of the graphs, we showed that searching was $O(1)$ for a single separator and $O(|V|)$ in total. This experiment seems to confirm that this assumption is fitting for classical input, as is the case with the triangle meshes of the Groningen Skeletonization Benchmark.

In Table 2 we show an excerpt of the running time measurements, including measurements of the phases of the algorithm. Here the vast gap in performance is clear, especially for `dragon.ply`, which is the largest input for which we have been able to run LSS, given a time frame of 20 hours. For this particular instance we achieve a running time that is almost a thousand times faster.

Of note is that LEMTS spends more time on projection, as expected, but less time on packing than LEM. As stated previously, the search for separators is often the dominating phase, however there are types of input for which this is not the case, as evidenced by `19465.ply`. The mesh consists of flat sheets with small details engraved, as can be seen in Figure 11. For both LSS and the multilevel algorithms, a large portion of the time is spent on packing and projection. This can occur if the separators are generally small and plentiful, so that many of them may quickly be found. For `19465.ply` these are particularly present around the imprinted text on the top sheets. It is worth noting that this would likely also be an example for which the structure of the input matters greatly for the running time of our multilevel algorithms.

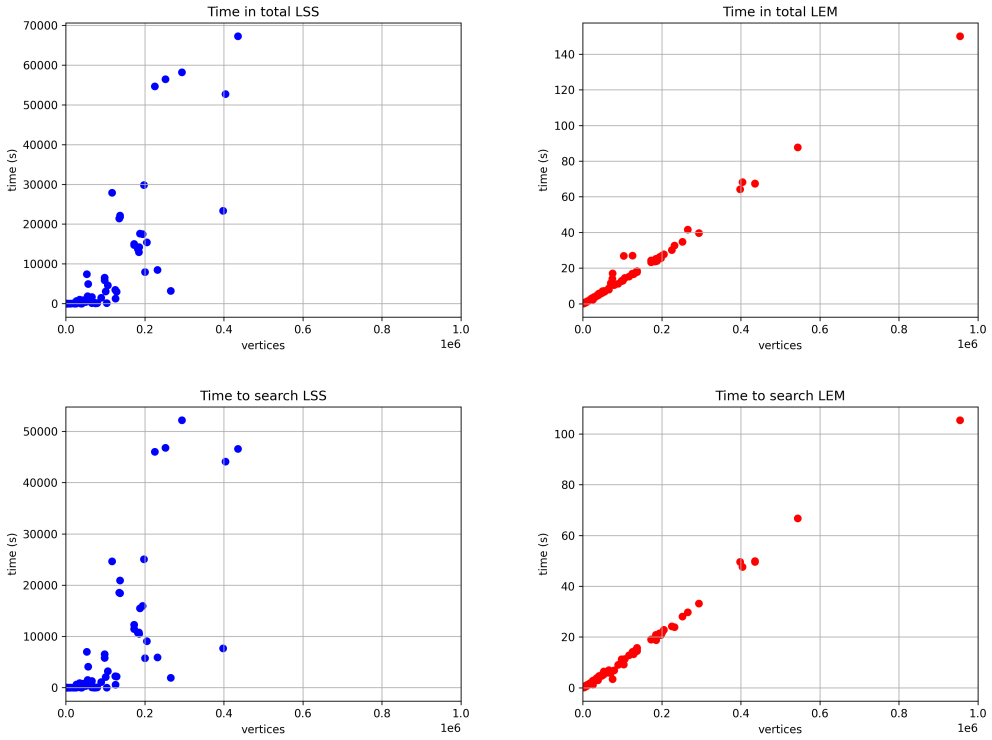
4 Conclusions and Future Work

We have proposed a multilevel algorithm for computing local separator-based curve skeletons, and shown that the approach is very efficient. We obtain a practical running time that appears near linear in the number of vertices of the input (see Figure 10) with up to thousandfold improvement in running time while not deteriorating the quality of the output substantially, if at all.

This type of running time improvement makes separator-based skeletonization applicable as a tool in biomedical image analysis, including frame-by-frame skeletonization of videos [3].

The application to video skeletonization motivates an unexplored line of related work, namely that of efficiently dynamically updating skeletons in a series of related shapes.

The multilevel approach offers great flexibility that has yet to be explored. It is easy to imagine coarsening schemes targeting specific structures of input, such as contracting clusters, rather than edges, on voxel input. These contraction schemes may provide new trade-offs between practical performance and skeleton quality.



(a) Running times of LSS. **(b)** Running times of LEM.

Figure 10 Running times in total (top) and for searching (bottom) as function of the number of vertices on the Groningen Skeletonization Benchmark. Values over 20 hours omitted.

Table 2 Excerpt of running time measurements. In addition to measuring the total time, we also measure the time spent on each phase of the algorithm.

input	algorithm	coarsen (s)	search (s)	project (s)	pack (s)	total (s)
19465	LSS	-	25.8787	-	138.836	164.714
	LEM	1.20002	9.18989	9.410914	12.4392	26.8306
	LEMTS	1.21294	9.41457	14.851733	9.8696	26.5518
dragon	LSS	-	46615.4	-	20637.9	67255.9
	LEM	6.25851	49.6567	18.753684	3.42042	67.4554
	LEMTS	6.19801	50.2347	24.6420235	3.22507	69.4672
fertility	LSS	-	95.8574	-	42.1409	136.664
	LEM	0.234681	2.38572	0.7753682	0.0600592	3.03186
	LEMTS	0.244822	2.4305	1.8177949	0.0559833	3.44687
happy4	LSS	-	7646.21	-	15731.9	23379.2
	LEM	6.89638	49.6684	12.7328477	2.22244	64.2245
	LEMTS	6.93433	50.1269	16.0581559	1.75872	65.4557
horse	LSS	-	544.182	-	85.6915	628.3
	LEM	0.504898	5.10887	1.2762728	0.0669417	6.27845
	LEMTS	0.502791	5.17779	2.7604771	0.0542464	6.86374
neptune	LSS	-	56.296	-	62.8555	119.232
	LEM	0.312728	2.7382	1.0849813	0.230939	3.77785
	LEMTS	0.308124	2.76549	2.4149765	0.182575	4.22093

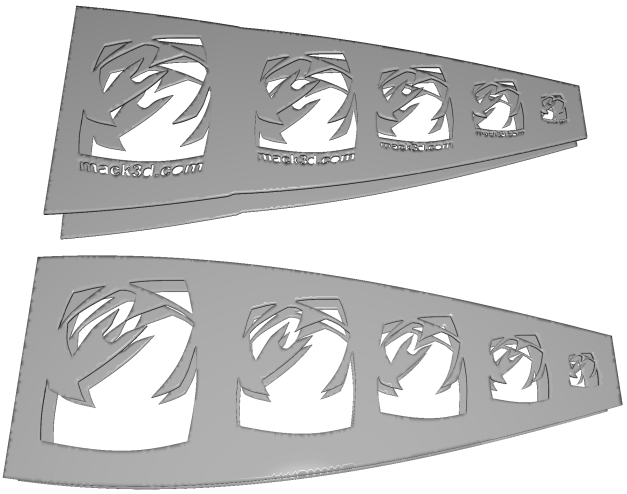


Figure 11 The triangle mesh `19465.ply`, where packing makes up a large portion of the running time for (all) local separator based skeletonization algorithms.

When applying coarsening to scale-free graphs, as might be the case for data visualisation or areas of application that are not classical for skeletonization, we move into a domain known from the field of graph partitioning to cause trouble for matching contraction schemes [1]. It is interesting to see if the improved practical performance, and the applicability to any spatially embedded graph, opens up for new areas of application of skeletons.

References

1 Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*. IEEE, 2006. doi:10.1109/IPDPS.2006.1639360.

2 Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In David C. Anderson and Kunwoo Lee, editors, *Sixth ACM Symposium on Solid Modeling and Applications, Sheraton Inn, Ann Arbor, Michigan, USA, June 4-8, 2001*, pages 249–266. ACM, 2001. doi:10.1145/376957.376986.

3 Kasra Arnavaz, Oswin Krause, Kilian Zepf, Jakob Andreas Bærentzen, Jelena M. Krivokapic, Silja Heilmann, Pia Nyeng, and Aasa Feragen. Quantifying topology in pancreatic tubular networks from live imaging 3d microscopy. *Machine Learning for Biomedical Imaging*, 1, 2022. URL: https://melba-journal.org/papers/2022:015.html.

4 Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):44, 2008. doi:10.1145/1360612.1360643.

5 Andreas Bærentzen and Eva Rotenberg. Skeletonization via local separators. *ACM Trans. Graph.*, 40(5):187:1–187:18, 2021. doi:10.1145/3459233.

6 Silvia Biasotti, Leila De Floriani, Bianca Falcidieno, Patrizio Frosini, Daniela Giorgi, Claudia Landi, Laura Papaleo, and Michela Spagnuolo. Describing shapes by geometrical-topological properties of real functions. *ACM Comput. Surv.*, 40(4):12:1–12:87, 2008. doi:10.1145/1391729.1391731.

7 Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Comput. Sci.*, 392(1-3):5–22, 2008. doi:10.1016/j.tcs.2007.10.018.

8 Angela Brennecke and Tobias Isenberg. 3d shape matching using skeleton graphs. In Thomas Schulze, Stefan Schlechtweg, and Volkmar Hinz, editors, *Simulation und Visualisierung 2004 (SimVis 2004) 4-5 März 2004, Magdeburg*, pages 299–310. SCS Publishing House e.V., 2004. URL: http://wwwisg.cs.uni-magdeburg.de/graphik/pub/files/Brennecke_2004_3SM.pdf.

9 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer International Publishing, 2016. doi:10.1007/978-3-319-49487-6_4.

10 J. Andreas Bærentzen. Gel. https://github.com/janba/GEL, 2022.

11 Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhixun Su. Point cloud skeletons via laplacian based contraction. In *SMI 2010, Shape Modeling International Conference, Aix en Provence, France, June 21-23 2010*, pages 187–197. IEEE Computer Society, 2010. doi:10.1109/SMI.2010.25.

12 Nicu D. Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Trans. Vis. Comput. Graph.*, 13(3):530–548, 2007. doi:10.1109/TVCG.2007.1002.

13 Tamal K. Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. In Alla Sheffer and Konrad Polthier, editors, *Proceedings of the Fourth Eurographics Symposium on Geometry Processing, Cagliari, Sardinia, Italy, June 26-28, 2006*, volume 256 of *ACM International Conference Proceeding Series*, pages 143–152. Eurographics Association, 2006. doi:10.2312/SGP/SGP06/143-152.

14 William Harvey, Yusu Wang, and Rephael Wenger. A randomized $O(m \log m)$ time algorithm for computing reeb graphs of arbitrary simplicial complexes. In David G. Kirkpatrick and Joseph S. B. Mitchell, editors, *Proceedings of the 26th ACM Symposium on Computational Geometry, Snowbird, Utah, USA, June 13-16, 2010*, pages 267–276. ACM, 2010. doi:10.1145/1810959.1811005.

15 M. Sabry Hassouna and Aly A. Farag. Variational curve skeletons using gradient vector flow. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2257–2274, 2009. doi:10.1109/TPAMI.2008.271.

16 Bruce Hendrickson and Robert W. Leland. A multi-level algorithm for partitioning graphs. In Sidney Karin, editor, *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, page 28. ACM, 1995. doi:10.1145/224170.224228.

17 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

18 Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L_1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65:1–65:8, 2013. doi:10.1145/2461912.2461913.

19 Andrei C. Jalba, Jacek Kustra, and Alexandru C. Telea. Surface and curve skeletonization of large 3d models on the GPU. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(6):1495–1508, 2013. doi:10.1109/TPAMI.2012.212.

20 Wei Jiang, Kai Xu, Zhi-Quan Cheng, Ralph R. Martin, and Gang Dang. Curve skeleton extraction by coupled graph contraction and surface clustering. *Graph. Model.*, 75(3):137–148, 2013. doi:10.1016/j.gmod.2012.10.005.

21 George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In Sidney Karin, editor, *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, page 29. ACM, 1995. doi:10.1145/224170.224229.

22 Mattia Natali, Silvia Biasotti, Giuseppe Patanè, and Bianca Falcidieno. Graph-based representations of point clouds. *Graph. Model.*, 73(5):151–164, 2011. doi:10.1016/j.gmod.2011.03.002.

23 Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007. doi:10.1145/1276377.1276449.

24 Diane Perchet, Catalin I. Fetita, and Françoise J. Prêteux. Advanced navigation tools for virtual bronchoscopy. In Edward R. Dougherty, Jaakko Astola, and Karen O. Egiazarian, editors, *Image Processing: Algorithms and Systems III, San Jose, California, USA, January 18, 2004*, volume 5298 of *SPIE Proceedings*, pages 147–158. SPIE, 2004. doi:10.1117/12.533096.

25 Dennie Reniers, Jarke J. van Wijk, and Alexandru C. Telea. Computing multiscale curve and surface skeletons of genus 0 shapes using a global importance measure. *IEEE Trans. Vis. Comput. Graph.*, 14(2):355–368, 2008. doi:10.1109/TVCG.2008.23.

26 Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. *Comput. Graph. Forum*, 31(5):1735–1744, 2012. doi:10.1111/j.1467-8659.2012.03178.x.

27 Andrea Tagliasacchi, Thomas Delamé, Michela Spagnuolo, Nina Amenta, and Alexandru C. Telea. 3d skeletons: A state-of-the-art report. *Comput. Graph. Forum*, 35(2):573–597, 2016. doi:10.1111/cgf.12865.

28 Alexandru C. Telea. 3d skeletonization benchmark. https://webspace.science.uu.nl/~telea001/Shapes/SkelBenchmark, 2016. Accessed: 2022-10-31.

29 Alexandru C. Telea and Andrei C. Jalba. Computing curve skeletons from medial surfaces of 3d shapes. In Hamish A. Carr and Silvester Czanner, editors, *Theory and Practice of Computer Graphics, Rutherford, United Kingdom, 2012. Proceedings*, pages 99–106. Eurographics Association, 2012. doi:10.2312/LocalChapterEvents/TPCG/TPCG12/099-106.

30 Ming Wan, Frank Dachille, and Arie E. Kaufman. Distance-field-based skeletons for virtual navigation. In Thomas Ertl, Kenneth I. Joy, and Amitabh Varshney, editors, *12th IEEE Visualization Conference, IEEE Vis 2001, San Diego, CA, USA, October 24-26, 2001, Proceedings*, pages 239–246. IEEE Computer Society, 2001. doi:10.1109/VISUAL.2001.964517.

31 Yu-Shuen Wang and Tong-Yee Lee. Curve-skeleton extraction using iterative least squares optimization. *IEEE Trans. Vis. Comput. Graph.*, 14(4):926–936, 2008. doi:10.1109/TVCG.2008.38.

32 Yajie Yan, Kyle Sykes, Erin W. Chambers, David Letscher, and Tao Ju. Erosion thickness on medial axes of 3d shapes. *ACM Trans. Graph.*, 35(4):38:1–38:12, 2016. doi:10.1145/2897824.2925938.